Social Networks III
Support Vector Machines I

Alexander Schönhuth

UNIVERSITÄT BIELEFELD
Faculty of Technology

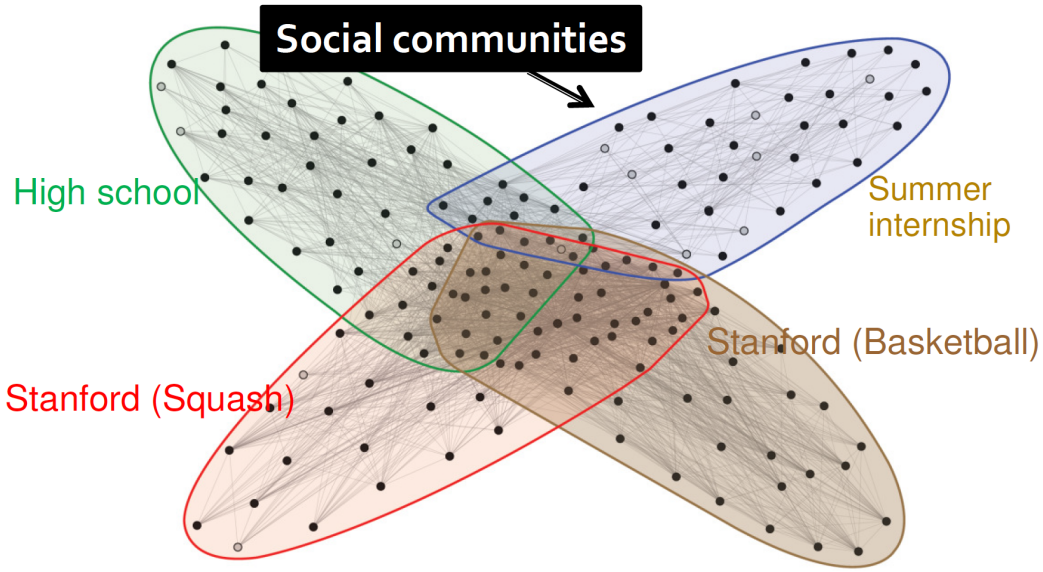
Bielefeld University
July 5, 2023

Learning Goals Today / Overview

- Overlapping communities: the Graph Affiliation Model
- Direct discovery of overlapping communities
- Supervised learning: summary

Reminder: Graph Affiliation Model

Overlapping Communities

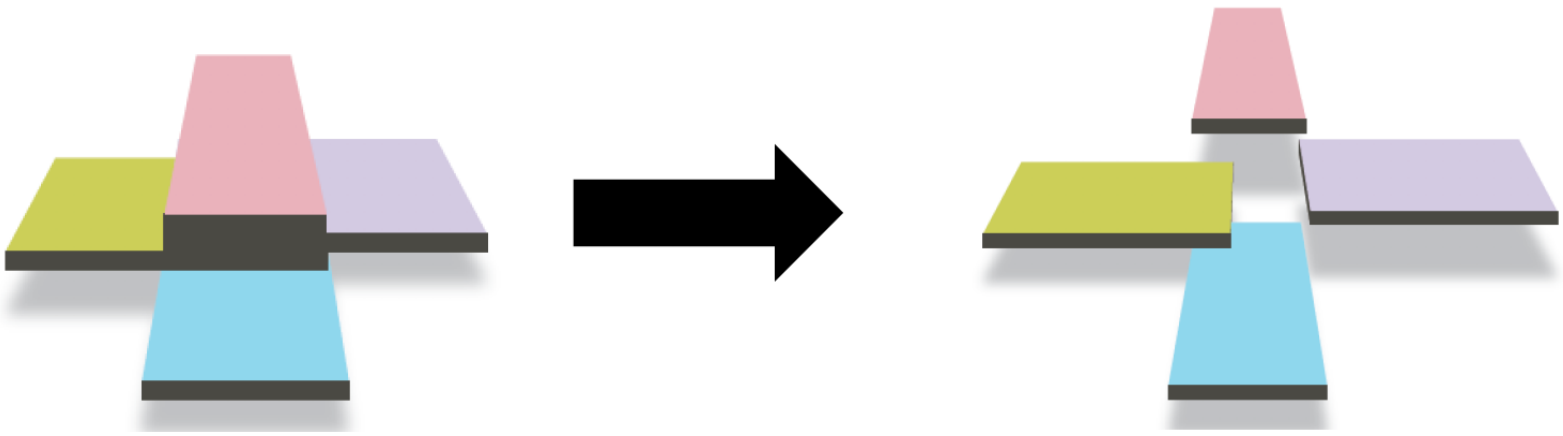


Subgraph from Facebook

Adopted from `mmds.org`

- *Observation:* Communities in social networks can overlap
- Graph partitioning does not help in these cases
- Would like to have a statistical interpretation of network data

Community Discovery: Goal



Revealing (overlapping) communities

Adopted from `mmds.org`

- *Goal:* Discover communities correctly
- Regardless of whether they overlap or not

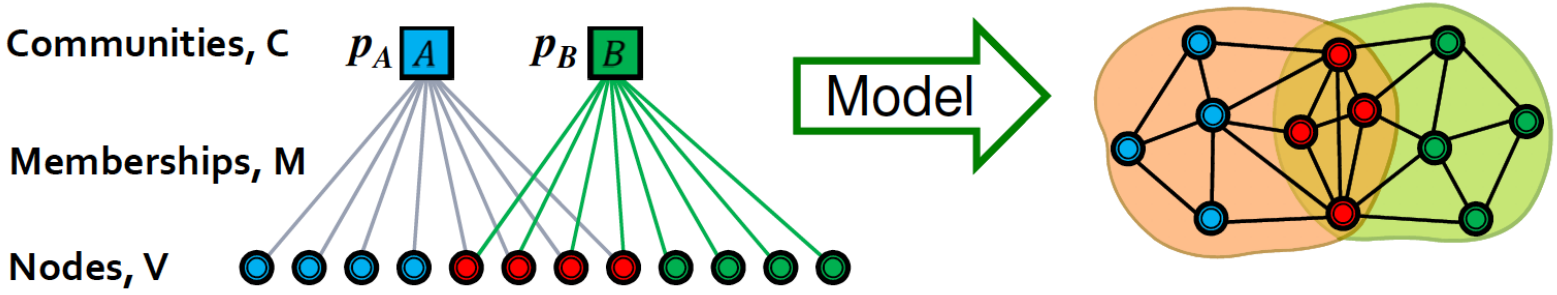
Determine the statistically most plausible community structure

Affiliation Graph Model: Introduction

- *Issue:* Statistical control over community structure of a network
- *Idea:* Design *generative probability distribution*
- Given a number of nodes, this generative distribution generates edges
- The generative distribution represents a particular community structure
  - The distribution knows about nodes belonging to communities
  - It generates more edges within communities
  - It generates less edges between communities

Affiliation Graph Model: Introduction

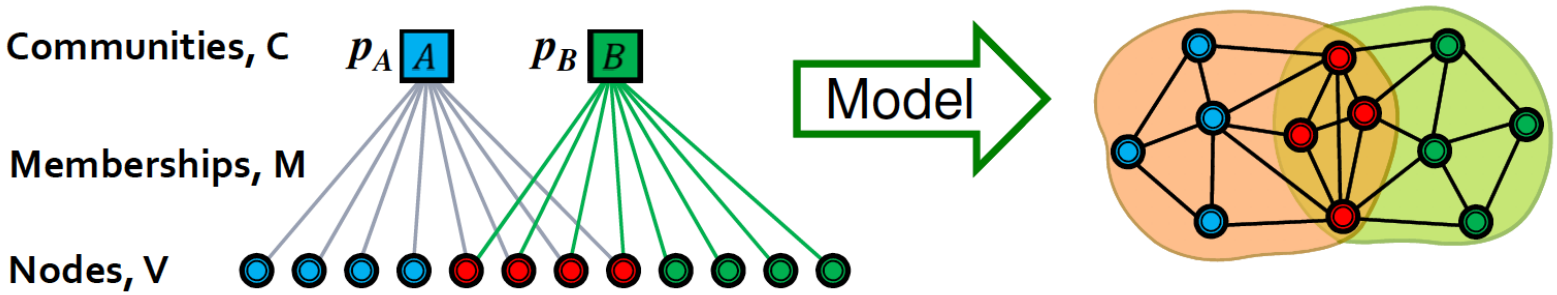
- *Issue:* Statistical control over community structure of a network
- *Idea:* Design *generative probability distribution*
- Given a number of nodes, this generative distribution generates edges



Distribution representing a community structure generating network

Adopted from `mmds.org`

Affiliation Graph Model: Introduction



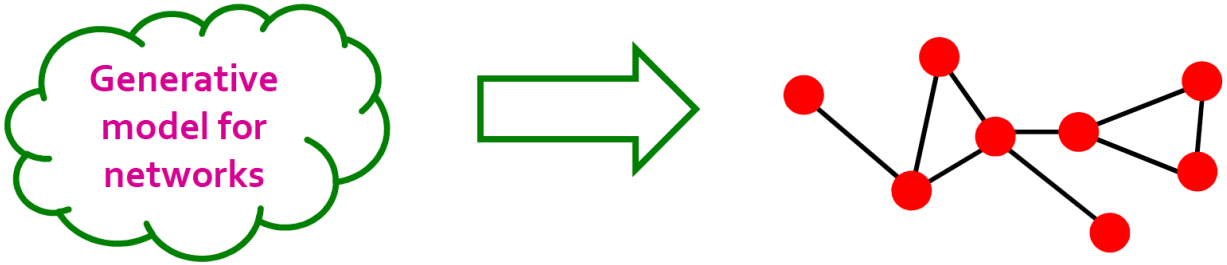
Distribution representing a community structure (left) generating network (right)

Adopted from `mmds.org`

- We can generate networks when knowing community structure
- *But:* We would like to determine the community structure when knowing the network

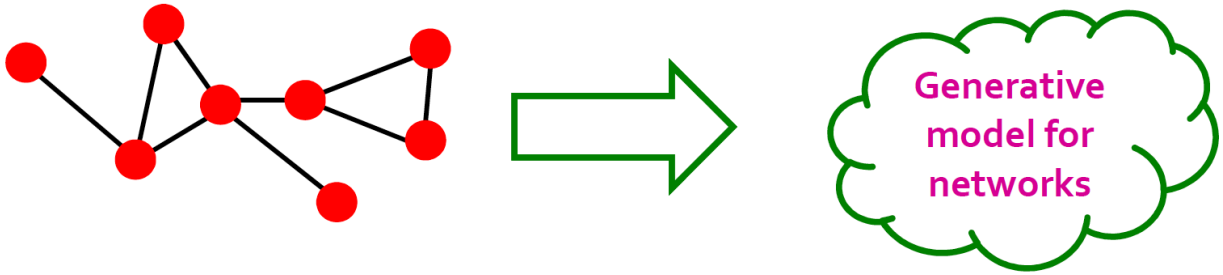
Isn't that exactly the opposite?

Generative Distributions



We can do this: generating network from distribution...

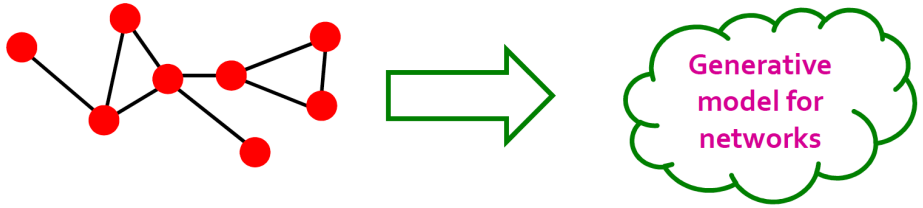
Adopted from `mmds.org`



...but we want this: inferring distribution from network

Adopted from `mmds.org`

Generative Distributions: Maximum Likelihood Inference



We want to infer distribution from network

Adopted from `mmds.org`

Maximum Likelihood Estimation

- Let $\mathbf{P}(N \mid \theta)$ be the probability that distribution $\theta \in \Theta$ generates network N
- *Maximum likelihood estimation:* Determine distribution $\hat{\theta}$ that generated N with greatest likelihood:

$$\hat{\theta} := \arg\max_{\theta \in \Theta} \mathbf{P}(N \mid \theta) \tag{1}$$

- This computes most reasonable distribution $\hat{\theta}$ for network N

Affiliation Graph Model: Definition I

- An AGM θ generates a network $N = (V, E)$ by adding edges E to a given set of nodes V
- For $u, v \in V$, edge (u, v) is generated with probability $\mathbf{P}_\theta((u, v))$
- $\mathbf{P}_\theta((u, v))$ depends on the parameters θ
- Recall that θ specifies community structure

So, what exactly is θ supposed to be?

Affiliation Graph Model: Parameters

- $\mathcal{C}$, as a set of *communities*
- $M \in \{0,1\}^{\mathcal{C} \times V}$, specifying *assignment of nodes* $v \in V$ *to communities* $C \in \mathcal{C}$, where

$$M_{C,v} = \begin{cases} 1 & v \text{ belongs to } C \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

  - M specifies "affiliations" of nodes $v \in V$
  - Note that one can vary $\mathcal{C}$, as a parameter, but not V
- $(p_C)_{C \in \mathcal{C}}$ as probabilities to generate edges (u, v) because $u, v \in C$
- *Summary:* A particular AGM θ corresponds to

$$\theta = (\mathcal{C}, M, (p_C)_{C \in \mathcal{C}}) \tag{3}$$

Affiliation Graph Model: $\mathbf{P}_\theta((u,v))$

Several C containing both u, v

- Let $M_u, M_v \subset \mathcal{C}$ be the subsets of communities that contain u and v, respectively
- Existence of communities that contain both u, v means

$$M_u \cap M_v \neq \emptyset$$

- Memberships in different communities have no influence on each other
- That is, we assume *statistical independence*

Affiliation Graph Model: $\mathbf{P}_\theta((u,v))$

Several C containing both u, v

- Statistical independence is expressed by

$$\prod_{C \in M_u \cap M_v} (1 - p_C)$$

  as probability of *no edge* (u, v) *in any community* $C \in M_u \cap M_v$
- Hence, the probability to generate (u, v) is

$$1 - \prod_{C \in M_u \cap M_v} (1 - p_C) \tag{4}$$

Done? No: What about $M_u \cap M_v = \emptyset$?

Affiliation Graph Model: $\mathbf{P}_\theta((u,v))$

No C containing both u, v

- For $M_u \cap M_v = \emptyset$, computing (4) yields (empty product is 1)

$$1 - \prod_{C \in \emptyset} (1 - p_C) = 1 - 1 = 0$$

- No edges across communities makes no sense
- Let $\epsilon > 0$ be small; we generate an edge (u, v) with probability

$$\mathbf{P}_\theta((u,v)) = \epsilon \quad \text{if} \quad M_u \cap M_v = \emptyset$$

Affiliation Graph Model: $\mathbf{P}_\theta((u,v))$

Affiliation Graph Model (AGM)

- An edge (u,v) is generated with probability

$$\mathbf{P}_\theta((u,v)) = \begin{cases} 1 - \prod_{C \in M_u \cap M_v} (1 - p_C) & M_u \cap M_v \neq \emptyset \\ \epsilon & M_u \cap M_v = \emptyset \end{cases} \tag{5}$$

- Edges (u,v) are generated independently from one another
- *Overall:* The probability $\mathbf{P}_\theta(E)$ to generate edges E given AGM θ computes as

$$\mathbf{P}_\theta(E) = \prod_{(u,v) \in E} \mathbf{P}_\theta((u,v)) \times \prod_{(u,v) \notin E} 1 - \mathbf{P}_\theta((u,v)) \tag{6}$$

where $\mathbf{P}_\theta((u,v))$ are computed following (5), with $\theta = (\mathcal{C}, M, p_C)$ determining p_C and M_u, M_v and so on.

Affiliation Graph Model: Overall Probability

Affiliation Graph Model (AGM)

- The probability $\mathbf{P}_\theta(E)$ to generate E given θ is

$$\mathbf{P}_\theta(E) = \prod_{(u,v)\in E} \mathbf{P}_\theta((u,v)) \times \prod_{(u,v)\notin E} 1 - \mathbf{P}_\theta((u,v)) \tag{7}$$

- *Reminder:* For a given network $N = (V, E)$, the *goal* is to determine

$$\hat{\theta} := \arg\max_{\theta\in\Theta} \mathbf{P}_\theta(E)$$

- That is, we need to vary $\theta = (\mathcal{C}, M, p_C)$ until $\mathbf{P}_\theta(E)$ is maximal

How to *systematically vary* $\theta = (\mathcal{C}, M, p_C)$?

Computing the MLE $\hat{\theta}$

Issues

- Search space of combinations of
  - Communities $\mathcal{C}$,
  - Assignments of nodes to communities M, and
  - Probabilities p_C for communities

  tends to be huge
- Concise formulas of (7) for $\mathbf{P}_\theta(E)$ as function of θ too difficult
- Analytical solution for determining $\hat{\theta} := \arg\max_{\theta \in \Theta} \mathbf{P}_\theta(E)$ not available
- Moreover, parameters are both discrete $(\mathcal{C}, M)$ and continuous $((p_C)_{C \in \mathcal{C}})$

Computing the MLE $\hat{\theta}$

Approach

1. Pick initial set of parameters θ_0
2. Vary θ such that $\mathbf{P}_\theta(E)$ iteratively increases
3. Vary $\mathcal{C}$ or M first
   - ☞ Partial derivates of $\mathbf{P}_\theta(E)$ wrt. p_C computable on fixed $\mathcal{C}, M$
4. Determine optimal $(p_C)_{C \in \mathcal{C}}$, e.g. by gradient descent
5. Keep change if $\mathbf{P}_\theta(E)$ has increased, discard otherwise

Computing the MLE $\hat{\theta}$

Iterative variations of $\mathcal{C}, M$

- *Varying M:*
  - Delete node from community, i.e. for $M_{C,v} = 1$, set $M_{C,v} = 0$
  - Add node to community, i.e. for $M_{C,v} = 0$, set $M_{C,v} = 1$
- *Varying $\mathcal{C}$:*
  - Merge two communities
  - Split community
  - Delete community
  - Add new community, with initial random selection of members

Computing the MLE $\hat{\theta}$

Soft Community Membership

- Instead of $M_{C,v} \in \{0,1\}$, allow any real-numbered $M_{C,v} \geq 0$
- For (u,v) to be generated because of $u, v \in C$, let

$$\mathbf{P}_\theta((u,v)) = 1 - e^{-M_{C,u}M_{C,v}} \tag{8}$$

  be the individual probability
- Proceeding exactly as before, we obtain

$$\mathbf{P}_\theta(E) = \prod_{(u,v)\in E} (1 - e^{-\sum_C M_{C,u}M_{C,v}}) \prod_{(u,v)\notin E} e^{-\sum_C M_{C,u}M_{C,v}} \tag{9}$$

Computing the MLE $\hat{\theta}$

Soft Community Membership

- Probability for edges E:

$$\mathbf{P}_\theta(E) = \prod_{(u,v)\in E} (1 - e^{-\sum_C M_{C,u} M_{C,v}}) \prod_{(u,v)\notin E} e^{-\sum_C M_{C,u} M_{C,v}} \tag{10}$$

- On fixed communities, include M in gradient descent (or related) optimization step
- *Advantages:*
  - Only one gradient descent run necessary
  - Less prone to get stuck in unfavorable local optima
- If necessary, add or delete communities, and re-run

Direct Discovery of Overlapping Communities

Introduction

- *Popular idea:* Determine communities as (induced) subgraphs of a certain type
- Subgraphs should contain unusually large amount of edges
- Subgraphs are allowed to overlap
- Will treat two types briefly here:
  - Cliques
  - Complete bipartite subgraphs

Finding Cliques

Definition [Induced Subgraph]
Let $G = (V, E)$ be a graph. A subgraph $C = (V' \subset V, E' \subset E)$ is *induced* iff

$$(v', w') \in E \quad \text{implies} \quad (v', w') \in E'$$

for any $v', w' \in V'$.

Definition [Clique]
Let $G = (V, E)$ be a graph.

- An induced subgraph $C = (V', E')$ is called a *clique* iff any pair of nodes in C is connected by an edge.
- A clique $C = (V', E')$ is *maximal* iff extending the clique by any node and its edges implies that the clique property no longer holds.

Communities as Cliques

- *Possible idea:* Determine communities as maximal cliques
- *Caveat:* The number of maximal cliques in a graph may be exponential in the number of nodes
- So, listing all maximal cliques is a computationally demanding problem
- Nevertheless, identifying communities as clique like arrangements is popular

Complete Bipartite Graphs

Definition [(Complete) Bipartite Graphs]

A graph $G = (V, E)$ with vertices V and edges E is referred to as *bipartite* iff

- there are $V_1, V_2 \subset V$ such that

$$V = V_1 \dot{\cup} V_2 \quad \text{and} \quad E \subset (V_1 \times V_2)$$

- A bipartite graph $G = (V, E)$ is *complete* iff

$$V = V_1 \dot{\cup} V_2 \quad \text{and} \quad E = (V_1 \times V_2)$$

  that is iff each node from V_1 is connected with each node from V_2
- A complete bipartite graph where $|V_1| = s, |V_2| = t$ is referred to as $K_{s,t}$
- A complete bipartite graph is also referred to as *biclique*

Complete Bipartite Graphs and Communities

- *Strategy:* Seek to discover all sufficiently large bicliques
- Treat them as "nuclei" (or seeds) of communities
- *Theoretical Advantage over Cliques:* While it is not possible to guarantee the existence of large cliques for graphs with many edges, one can guarantee the existence of large bicliques

Finding Complete Bipartite Graphs

Frequent Itemset Mining Problem

- Let $G = (V, E)$ on $V = V_1 \dot{\cup} V_2$ be a (large) bipartite graph
- Items are nodes from V_1
- Baskets are nodes from V_2
- Items in baskets are nodes from V_1 connected to basket node
- $K_{s,t}$ in G is itemset of size s that appears in t baskets
- So mining for frequent itemsets at threshold t dicovers all $K_{s,t}$

Supervised Learning

Supervised Learning

- There is a functional relationship

$$f^* : \mathbb{R}^d \to V$$

  we would like to understand, or *learn*.
  - *Regression*: $V = \mathbb{R}$
  - *Classification*: $V = \{1, ..., k\}$
- To learn it, we are given *m data points*

$$(x_i, f^*(x_i) = y_i)_{i=1,...,m}$$

  that reflect this functional relationship.

Final goal: Predict $f^*(x)$ well on unknown data points x.

Supervised versus Unsupervised Learning

- *Unsupervised Learning:*
  - Given unlabeled data

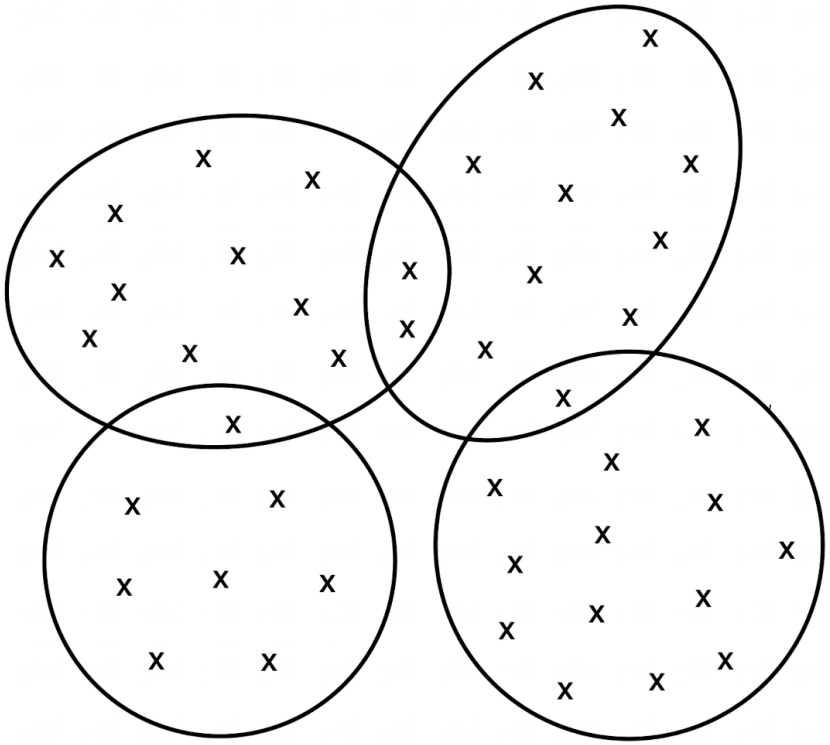
$$(x_i)_{i=1,\ldots,m}$$

  - *Goal:* Infer subgroups of data points
  - *Alternative Problem Formulation:* Learn the probability distribution

$$\mathbf{P}(\mathbf{X})$$

    that governs the generation of data points

Unsupervised Learning: Example

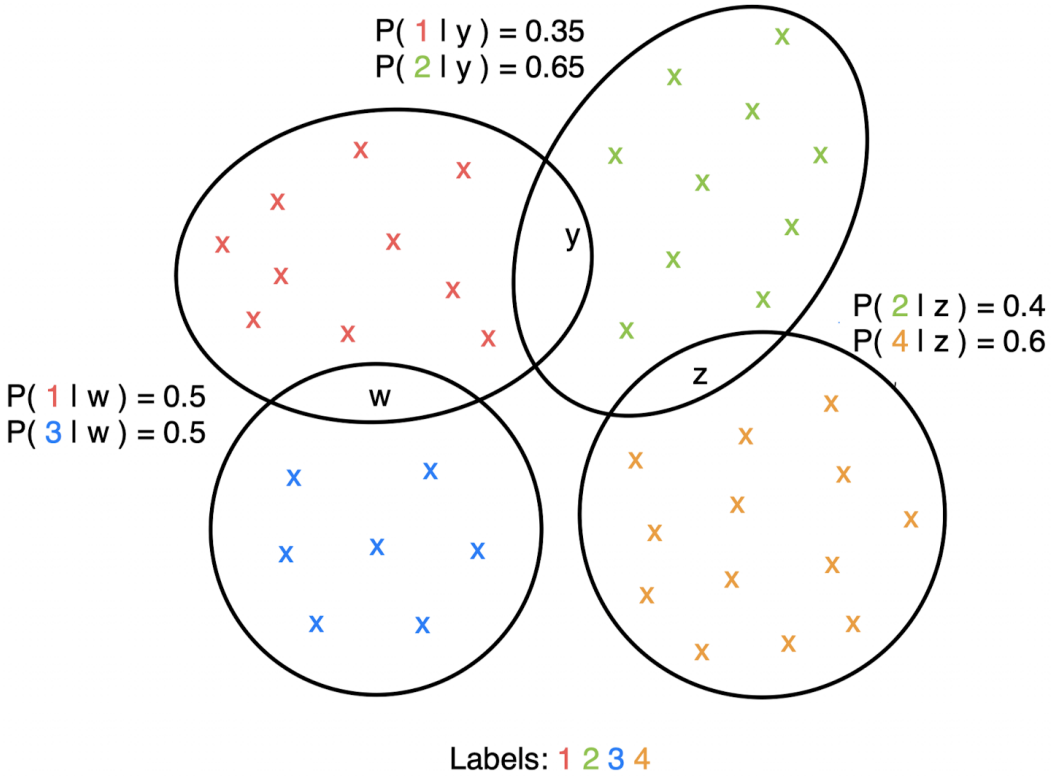


Generative distribution yielding four clusters

Supervised versus Unsupervised Learning

- *Supervised Learning:*
  - Given labeled data
    $$(x_i, y_i)_{i=1,\dots,m}$$
  - *Goal:* Learn functional relationship $f^* : \mathbb{R}^d \to V$, s.t. $y_i = f^*(x_i)$
  - *Alternative Problem Formulation:* Learn the probability distribution
    $$\mathbf{P}(\mathbf{X}, \mathbf{y}) \quad \text{or} \quad \mathbf{P}(\mathbf{y} \mid \mathbf{X})$$
    as a more general version of functional relationship

Unsuperviseed Learning: Example



Generative distribution yielding four clusters *and* corresponding labels

Supervised Learning: Training

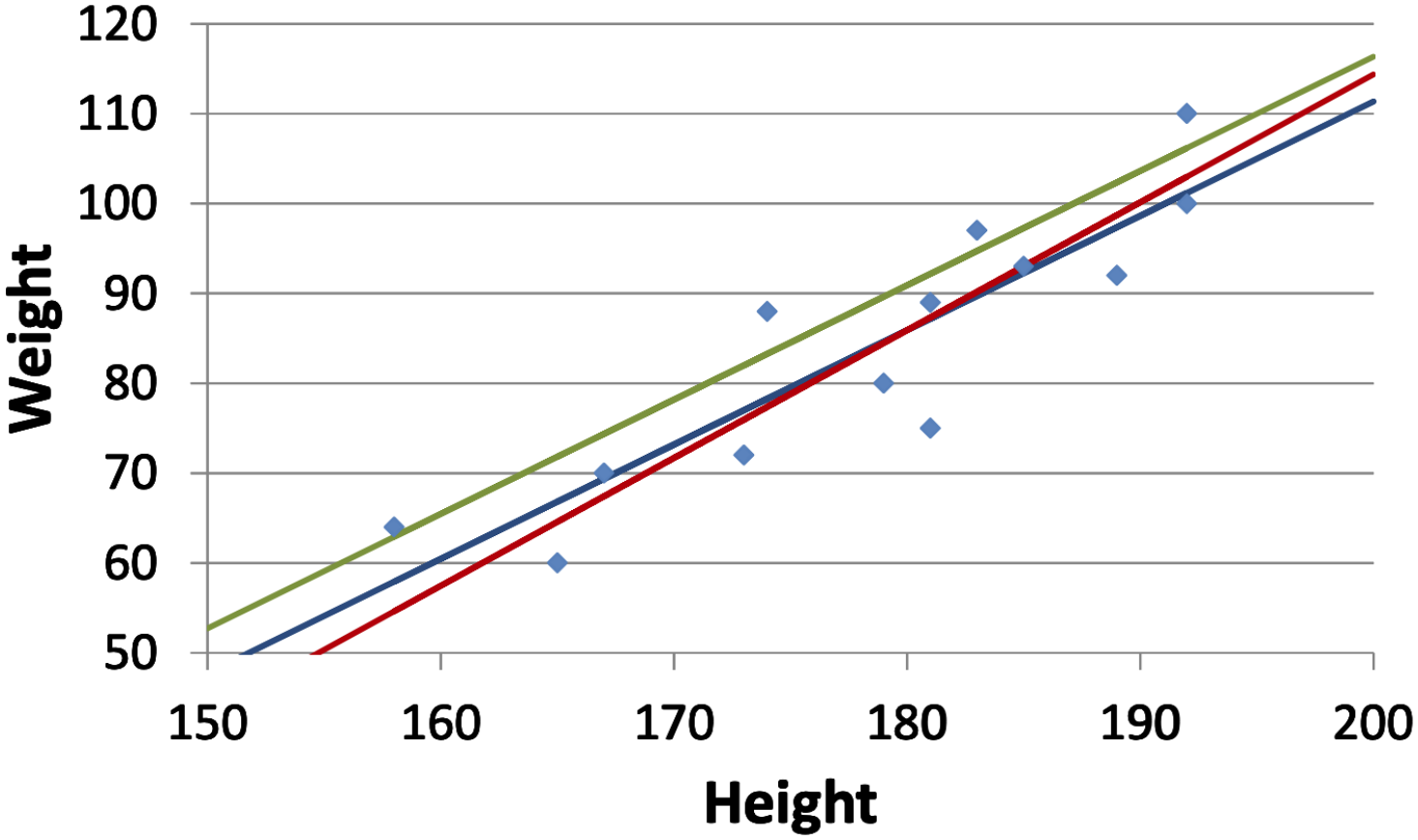
- The idea is to set up a *training procedure* (an algorithm) that *learns* f^* from the training data.
- Learning f^* means to *approximate* it by $f : \mathbb{R}^d \to V$ sufficiently well, where $f \in \mathcal{M}$ for a certain class of functions $\mathcal{M}$.
- In most cases, $f \in \mathcal{M}$ are parameterized by parameters $\mathbf{w}$. This means that we have to pick an appropriate choice of parameters $\mathbf{w}$ for learning f^*.

Supervised Learning

- We need to determine a *cost (or loss) function* C where $C(f, f^*)$ measures how well $f \in \mathcal{M}$ approximates f^*.
- *Optimization*: Pick $f \in \mathcal{M}$ (by picking the right set of parameters) that yields small (possibly minimal) cost $C(f, f^*)$
- *Generalization*: Optimization procedure should address that f is to approximate f^* well on *unknown data points*.

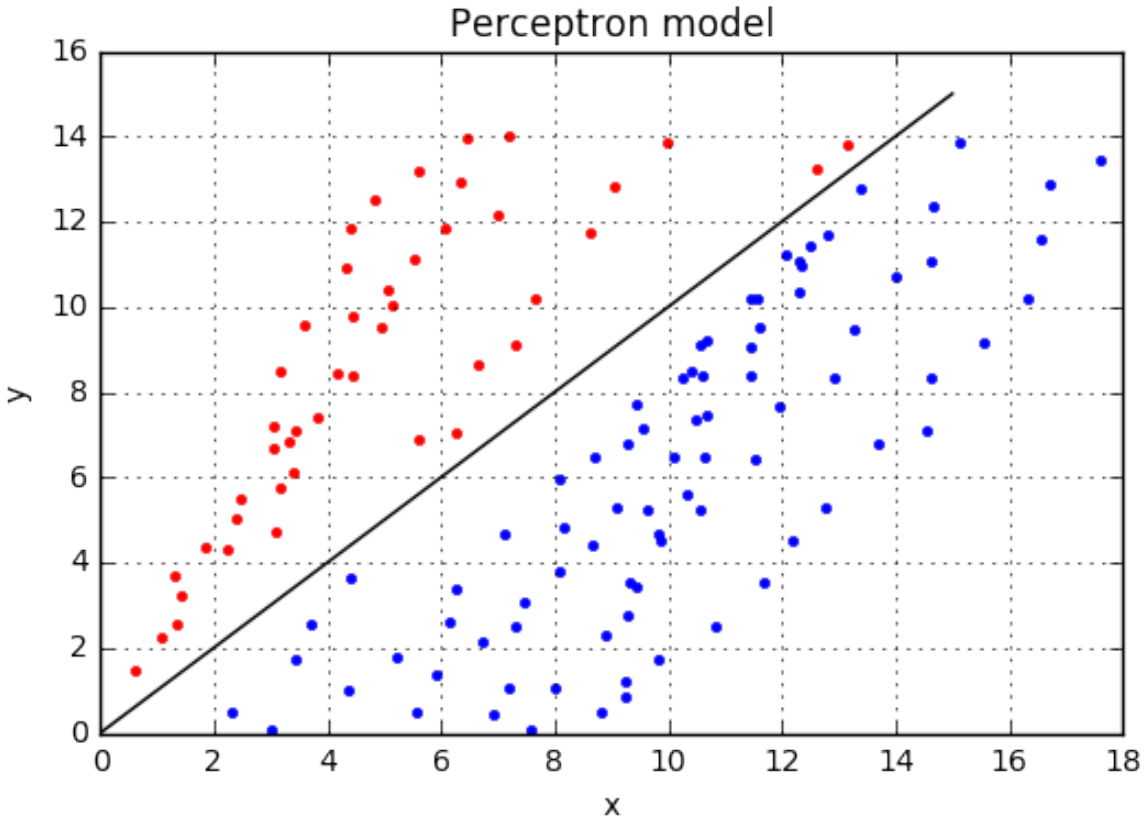
Linear Regression

Example: $f : \mathbb{R} \to \mathbb{R}$



Perceptron

Example: $f : \mathbb{R}^2 \to \{0, 1\}$



$$
\begin{aligned}
f \qquad \mathbb{R}^2 \quad &\longrightarrow \quad \{0 = \text{blue}, 1 = \text{red}\} \\
(x_1, x_2) \quad &\mapsto \quad \begin{cases} 1 & x_2 - x_1 > 0 \\ 0 & x_2 - x_1 \leq 0 \end{cases}
\end{aligned}
\tag{11}
$$

Supervised Learning

Summary

We need to specify:

- How to set up the data being used for training
- A model class $\mathcal{M}$, for example linear functions
- A cost function $C(f, f^*)$ that evaluates the goodness of $f \in \mathcal{M}$
- An optimization procedure that picks f such that $C(f, f^*)$ is minimal, or very small
- Keep in mind that f is to perform well on previously unseen data

Supervised Learning

Notation

- The dataset is given by a *design matrix* $\mathbf{X} \in \mathbb{R}^{m \times d}$ where m is the number of data points and d is the number of *features*
- Each data point x_i (a row in $\mathbf{X}$) is assigned to a *label* y_i that reflects the true functional relationship $y_i = f^*(x_i)$, where further $\mathbf{y} = (y_1, ..., y_m) \in V^m$ is the *label vector*.

Generalization

Enabling Generalization: Data

Training, Test and Validation

- Split $(\mathbf{X}, \mathbf{y})$ into
  - training data $(\mathbf{X}^{(\text{train})}, \mathbf{y}^{(\text{train})})$
  - validation data $(\mathbf{X}^{(\text{val})}, \mathbf{y}^{(\text{val})})$
  - test data $(\mathbf{X}^{(\text{test})}, \mathbf{y}^{(\text{test})})$
- *Training data:*
  - Used to pick the optimal set of parameters
  - That is, pick the optimal, particular element of $\mathcal{M}$
  - Training reflects common optimization procedure

Enabling Generalization: Data

Training, Test and Validation

- Split $(\mathbf{X}, \mathbf{y})$ into
  - training data $(\mathbf{X}^{(\text{train})}, \mathbf{y}^{(\text{train})})$
  - validation data $(\mathbf{X}^{(\text{val})}, \mathbf{y}^{(\text{val})})$
  - test data $(\mathbf{X}^{(\text{test})}, \mathbf{y}^{(\text{test})})$
- *Validation data:*
  - Used to determine *hyperparameters*
  - Hyperparameters refer to number of training iterations, choosing optimization procecure, neural network architecture variants
  - Some reflect selecting appropriate subsets of $\mathcal{M}$

Enabling Generalization: Data

Training, Test and Validation

- Split $(\mathbf{X}, \mathbf{y})$ into
  - training data $(\mathbf{X}^{(\text{train})}, \mathbf{y}^{(\text{train})})$
  - validation data $(\mathbf{X}^{(\text{val})}, \mathbf{y}^{(\text{val})})$
  - test data $(\mathbf{X}^{(\text{test})}, \mathbf{y}^{(\text{test})})$
- *Nested training cycle:*
  1. Train on training data using current hyperparameters
     ☞ Yields parameters
  2. Evaluate determined parameters on validation data
     ☞ Adjusting hyperparameters yields new hyperparameters
  3. Return to 1.
- Nested training yields optimal parameters and hyperparameters

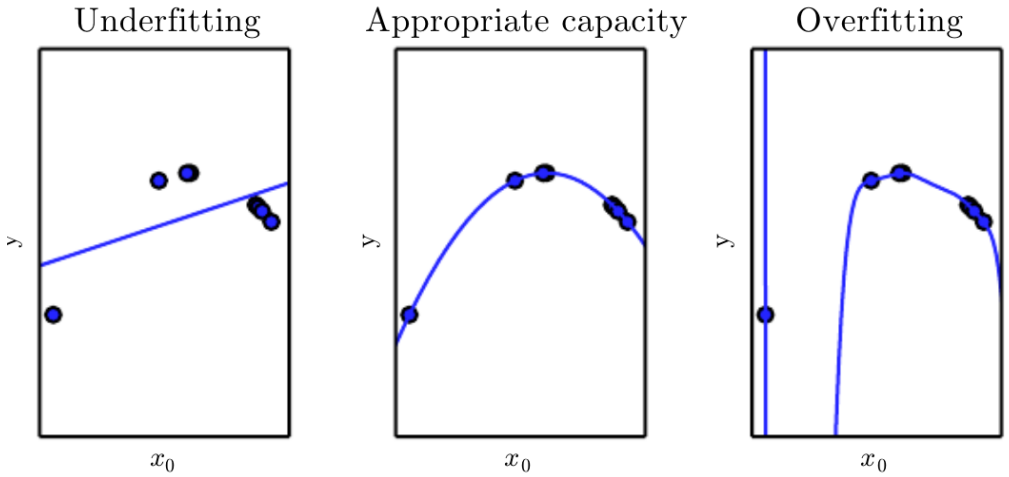
Enabling Generalization: Data

Training, Test and Validation

- Split $(\mathbf{X}, \mathbf{y})$ into
  - training data $(\mathbf{X}^{(\text{train})}, \mathbf{y}^{(\text{train})})$
  - validation data $(\mathbf{X}^{(\text{val})}, \mathbf{y}^{(\text{val})})$
  - test data $(\mathbf{X}^{(\text{test})}, \mathbf{y}^{(\text{test})})$
- *Test data:*
  - $(\mathbf{X}^{(\text{test})}, \mathbf{y}^{(\text{test})})$ are never touched during training
  - Final goal is to minimize cost on test data
- *Machine learning dilemma:* Optimize with respect to data you do not know

Enabling Generalization: Model

Capacity, Under- and Overfitting



Left: Linear functions underfit
Center: Polynomials of degree 2 neither under- nor overfit
Right: Polynomials of degree 9 overfit

- Choose a class of models that has the right *capacity*
- Capacity too large: *overfitting*
- Capacity too small: *underfitting*

Enabling Generalization: Cost Function

Regularization

Let $C(f, f^*)$ be the cost function. Let $\mathbf{w} = (w_1, ..., w_k)$ be the parameters specifying elements of $f_{\mathbf{w}} \in \mathcal{M}$.

- Usually, C refers to only known data points. That is, C evaluates as
$$C(f, f^*) = \sum_i C(f(x_i), y_i = f^*(x_i)) \tag{12}$$
where x_i runs over all training data points.
- Add a *regularization term* to cost function, and choose $f_{\mathbf{w}}$ that yields minimal
$$C(f_{\mathbf{w}}, f^*) + \lambda\Omega(\mathbf{w}) \tag{13}$$
- λ is a hyperparameter

Enabling Generalization: Cost Function

Regularization

- Prominent examples:
  - L_1 *norm*: $\Omega(\mathbf{w}) := \sum_i |w_i|$
  - L_2 *norm*: $\Omega(\mathbf{w}) := \sum_i w_i^2$
- Rationale: Penalize too many non-zero weights
- Virtually less complex model, hence virtually less capacity
- ☞ Prevents overfitting, yields better generalization

Enabling Generalization: Optimization

Early Stopping, Dropout

Optimization can be an iterative procedure.

- *Early stopping*: Stop the optimization procedure before cost function reaches an optimum on the training data.
- *Dropout*: Randomly fix parameters to zero, and optimize remaining parameters.

Enabling Generalization: Summary

- Training reflects an optimization procedure
- *Insight:* Optima correspond to overfitting training data
- *Solution:* Seek to output parameters "nearby" optima
- Nearly all generalization techniques address this:
  - Early stopping stops optimization before optimum is reached
  - Dropout carries out optimization in pre-set lower-dimensional subspace
  - Regularization forces to watch out for optima in lower-dimensional subspaces

Prominent Supervised Learning Model Examples

Linear Regression

- Design matrix $\mathbf{X} \in \mathbb{R}^{m \times d}$, label vector $\mathbf{y} \in \mathbb{R}^m$
- Model class: Let $\mathbf{w} \in \mathbb{R}^d$

$$f_{\mathbf{w}} = f(\mathbf{x}; \mathbf{w}) : \begin{array}{ccc} \mathbb{R}^d & \longrightarrow & \mathbb{R} \\ \mathbf{x} & \mapsto & \mathbf{w}^T \mathbf{x} \end{array} \tag{14}$$

- *Remark*: Note that the case $\mathbf{w}^T \mathbf{x} + b$ can be treated as a special case to be included in $\mathcal{M}$, by augmenting vectors $\mathbf{x}_i$ by an entry 1 (think about this...)
- Cost function (recall $y_i = f^*(\mathbf{x}_i)$)

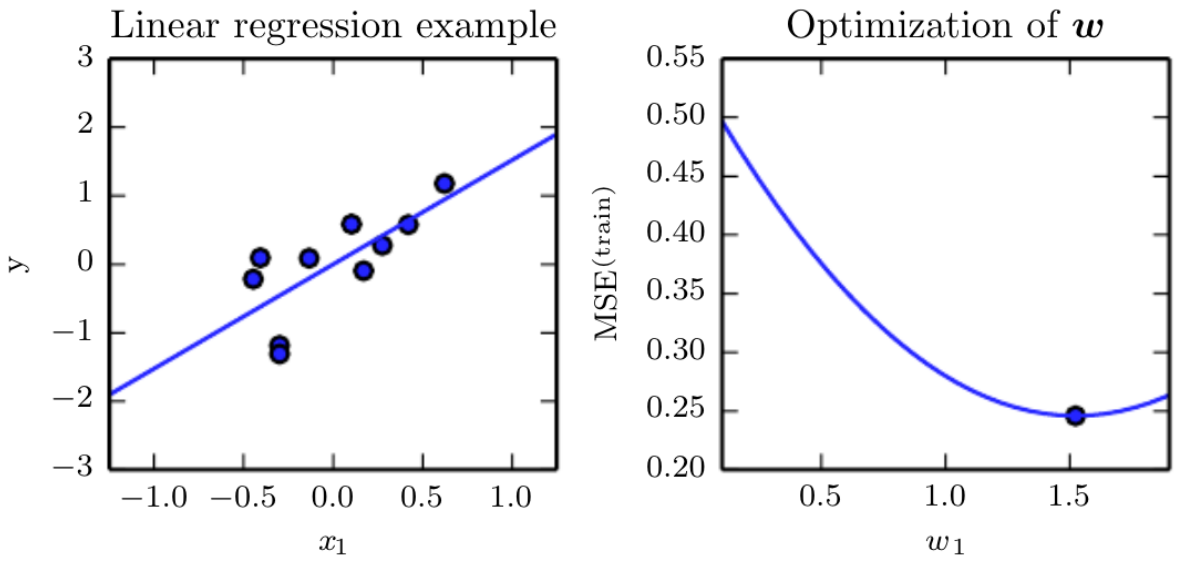
$$C(f, f^*) := \frac{1}{m} ||(f(\mathbf{x}_1), ..., f(\mathbf{x}_m)) - \mathbf{y}||_2^2 = \frac{1}{m} \sum_{i=1}^{m} (f(\mathbf{x}_i) - \mathbf{y}_i)^2 \tag{15}$$

Linear Regression

Optimization

- Solve for
$$\nabla_{\mathbf{w}} C(f_{\mathbf{w}}, f^*) = 0 \tag{16}$$
to achieve a minimum. This yields the *normal equations*
$$\mathbf{w} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y} \tag{17}$$
- *Global optimum* if $\mathbf{X}^T\mathbf{X}$ is invertible
- Do this on *training data* (so $\mathbf{X} = \mathbf{X}^{(\text{train})}, \mathbf{y} = \mathbf{y}^{(\text{train})}$) only. Hope that cost on test data is small.

Normal Equations



- *Left*: Data points, and the linear function $y = w_1 x$ that approximates them best
- *Right*: Mean squared error (MSE) depending on w_1
- *Remark on Perceptrons*: Optimizing is different, but also supported by a very easy optimization scheme (the *perceptron algorithm*)

Nearest Neighbor Classification

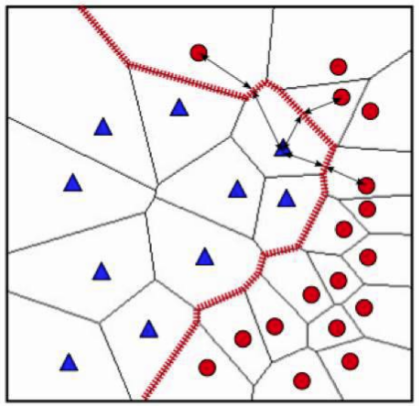
- Consider appropriate distance measure

$$D : \mathbb{R}^d \times \mathbb{R}^d \longrightarrow \mathbb{R}_+ \tag{18}$$

- For unknown data point $\mathbf{x}$, determine the closest given data point

$$\mathbf{x}_{i^*} := \operatorname{argmin}_i(D(\mathbf{x}, \mathbf{x}_i)) \tag{19}$$

- Predict label of $\mathbf{x}$ as y_{i^*}



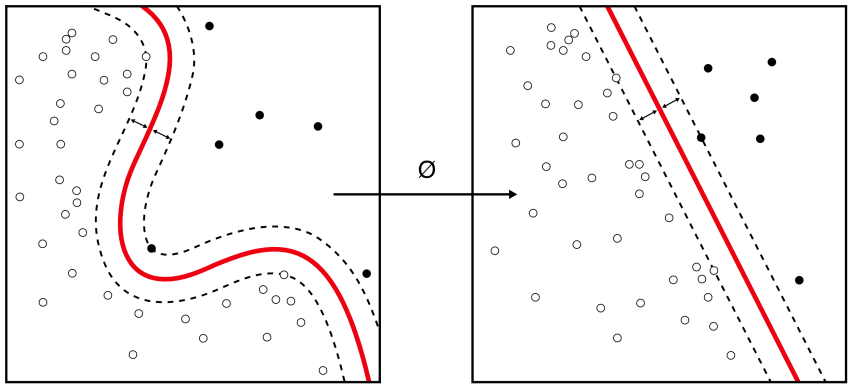
Support Vector Machines

- *Realization*: From (17), write

$$\mathbf{w}^T\mathbf{x} = \sum_{i=1}^{m} \alpha_i \mathbf{x}^T\mathbf{x}_i = \sum_{i=1}^{m} \alpha_i \langle \mathbf{x}, \mathbf{x}_i \rangle \tag{20}$$

- Replace $\langle ., . \rangle$ by different *kernel* (i.e. scalar product) $k(., .)$, that is by computing $\langle \phi(.), \phi(.) \rangle$ for appropriate ϕ

☞ Seek α's to maximize margin: still easy to optimize both for regression and classification!



General / Further Reading

Literature

- Mining Massive Datasets, Sections 10.3, 10.5, 12.1–12.3
  `http://infolab.stanford.edu/~ullman/mmds/ch10.pdf`